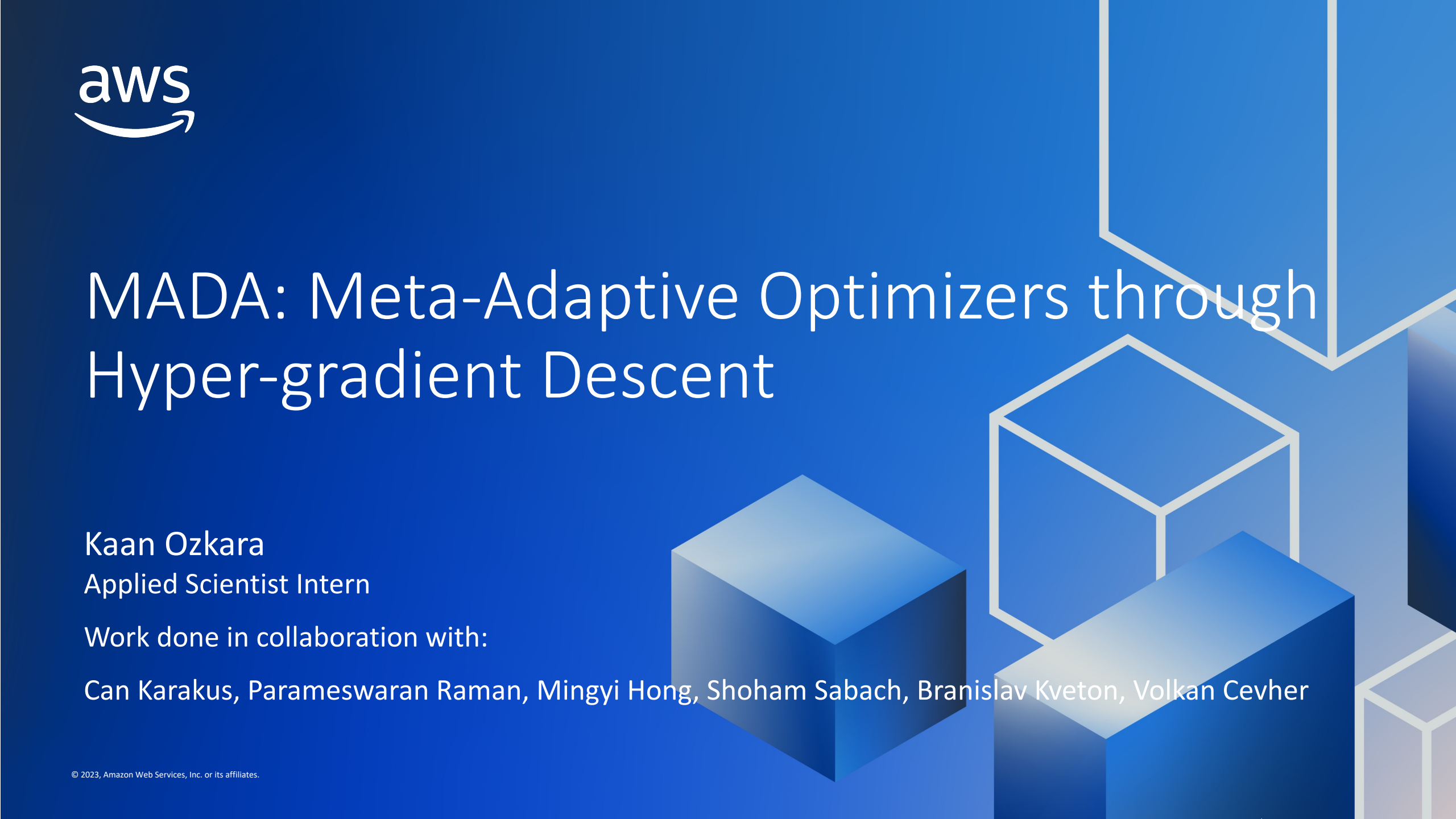


MADA: Meta-Adaptive Optimizers through Hyper-gradient Descent

Kaan Ozkara
Applied Scientist Intern

Work done in collaboration with:

Can Karakus, Parameswaran Raman, Mingyi Hong, Shoham Sabach, Branislav Kveton, Volkan Cevher

Outline

1. Background on optimizers
2. Parameterized optimizers
3. MADA: Meta-Adaptive Optimizers
4. AVGrad
5. Experiments
6. Future work

Problem

$$\min_{x \in R^d} F(x)$$

- F is differentiable.
- There exists $f: R^d \rightarrow R$ a random function computed on minibatch sampled from underlying data distribution such that $E[f(x)] = F(x)$ for all x.
- First order optimization methods are the workhorse in modern deep learning.

Overview of Adaptive moment optimizers

$$
\begin{aligned}
g_t &= \nabla f(x_t), \\
m_{t+1} &= \beta_1 m_t + (1-\beta_1)\,\tilde{g}_t, \\
x_{t+1} &= x_t - s_t \gamma_t \tilde{m}_{t+1},
\end{aligned}
$$

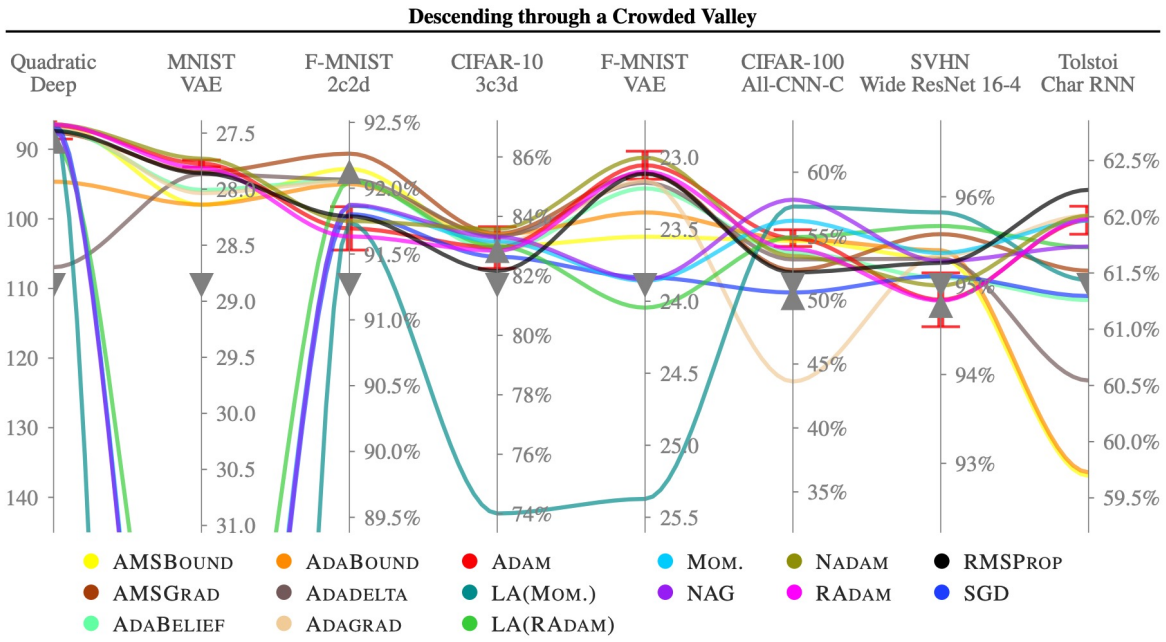
$$
\gamma_t = \begin{cases} \alpha_t, & \text{NADAM, ADAN, ADAM, YOGI} \\ 1, & \text{LANS, LARS, LAMB} \end{cases}
$$

$$
v_t = \begin{cases} \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2, & \text{NADAM, ADAM, LAMB} \\ v_{t-1} - (1-\beta_2)\, sign(v_{t-1} - g_t^2) g_t^2, & \text{YOGI} \\ \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 / \left\| g_t \right\|^2, & \text{LANS} \\ \beta_2 v_{t-1} + (1-\beta_2)\left(g_t + \beta_1 (g_t - g_{t-1})\right)^2, & \text{ADAN} \end{cases}
$$

$$
\tilde{m}_{t+1} = \begin{cases} m_{t+1}, & \text{ADAN, ADAM, YOGI} \\ \frac{\|x_t\|}{\|m_{t+1}\|} m_{t+1}, & \text{LARS} \\ \frac{\|x_t\|}{\|\alpha_t m_{t+1} + \lambda x_t\|} \left(\alpha_t m_{t+1} + \lambda x_t\right), & \text{LAMB} \\ \frac{\beta_1 \|x_t\| (\alpha_t m_{t+1} + \lambda x_t)}{\|\alpha_t m_{t+1} + \lambda x_t\|} + \frac{(1-\beta_1)\|x_t\|(\alpha_t g_t + \|g_t\| \lambda x_t)}{\|\alpha_t g_t + \|g_t\| \lambda x_t\|}, & \text{LANS} \\ \beta_1 m_{t+1} + (1-\beta_1)\, g_t, & \text{NADAM} \end{cases}
$$

$$
\tilde{g}_t = \begin{cases} g_t, & \text{NADAM, ADAM, YOGI, LAMB} \\ g_t + \lambda x_t, & \text{LARS} \\ \frac{g_t}{\|g_t\|}, & \text{LANS} \\ g_t + \beta_1 (g_t - g_{t-1}), & \text{ADAN} \end{cases}
$$

Motivation for a unified optimizer



(Schmidt et al., 2020)

Different optimizers perform better in different tasks, Adam remains the most popular esp. for LLMs.

A Unified Framework for Adaptive Optimizers

$$x_t = x_{t-1} - \alpha_t \frac{m_t}{\sqrt{v_t} + \epsilon},$$

Table 1 A unified framework to express adaptive moment optimizers.

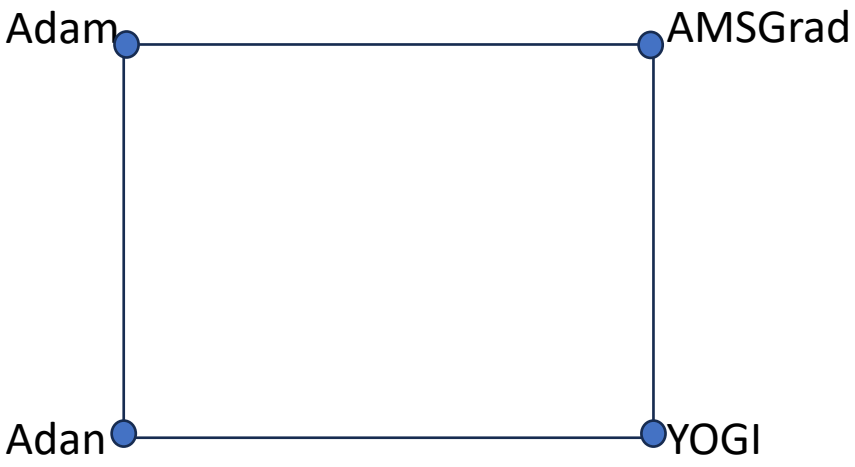
Method	First-Order Moment	Second-Order Moment
Adam [Kingma and Ba, 2015]	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
AMSGrad [Reddi et al., 2018]	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$\bar{v}_t = \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2$ $v_t = \max\{v_{t-1}, \bar{v}_t\}$
Adan [Xie et al., 2023]	$\bar{m}_t = \beta_1 \bar{m}_{t-1} + (1-\beta_1) g_t$ $n_t = \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1})$ $m_t = \bar{m}_t + \beta_3 n_t$	$\hat{g}_t = g_t + \beta_3 (g_t - g_{t-1})$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) \hat{g}_t^2$
Yogi [Zaheer et al., 2018]	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$\hat{g}_t = v_{t-1} + g_t^2 \cdot \mathrm{sign}(g_t^2 - v_{t-1})$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) \hat{g}_t$

Parameterized Optimizers

Existing Optimizers → Parameterization → Parameterized Optimizer

Informally, a parameterized optimizer can be described as the convex hull of a set of optimizers, when mapped to a Euclidean space under a certain parameterization.

We introduce interpolation coefficients that is restricted between 0 and 1.



Parameterized Optimizers

First order moment:

$$\bar{m}_t = \beta_1 \bar{m}_{t-1} + (1-\beta_1) g_t$$
$$n_t = \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1})$$
$$m_t = \bar{m}_t + \beta_3 n_t,$$

Second order moment:

$$\hat{g}_t = g_t + \beta_3 (g_t - g_{t-1})$$
$$\tilde{g}_t^2 = c\hat{g}_t^2 + (1-c)(v_{t-1} + \hat{g}_t^2 \cdot \mathrm{sign}(\hat{g}_t^2 - v_{t-1}))$$
$$\tilde{v}_t = \beta_2 \tilde{v}_{t-1} + (1-\beta_2)\tilde{g}_t^2$$
$$v_t^{(max)} = \max\{v_{t-1}^{(max)}, \tilde{v}_t\}$$
$$v_t = \rho \tilde{v}_t + (1-\rho) v_t^{(max)}.$$

Particular optimizers can be obtained through setting coefficients:

- ADAM: $\beta_3 = 0, c = 1, \rho = 1$
- Adan: $c = 1, \rho = 1$
- AMSGrad: $\beta_3 = 0, c = 1, \rho = 0$
- YOGI: $\beta_3 = 0, c = 0, \rho = 1$

Meta-Adaptive Optimizers

How to learn parameters of parameterized optimizers?

Algorithm 1 Pseudocode for a generic `MADA`

Input: A parameterized optimizer $\mathcal{O}_q$, where $q \in \mathcal{D}$, a hyper-learning rate α, number of total iterations T.
Init.: x_0 and q_0.

1: **for** t=1 to T **do**
2: Sample f_t.
3: Update the model parameters:

$$x_t = \mathcal{O}_{q_{t-1}}(x_{t-1}).$$

4: Update the optimizer coefficients:

$$q_t = \Pi_{\mathcal{D}} \left[q_{t-1} - \alpha \nabla_q f_t(x_{t-1}) \right].$$

5: **end for**

Output: Model wights x_T.

$\mathcal{O}_q$ denotes the parameterized optimizer with parameter set q, $\mathcal{D}$ denotes the domain of the parameters.

Hyper-gradient descent

- To compute the hyper-gradient of the loss with respect to a particular optimizer coefficient, we treat the updated model weights as a function of the coefficient (Baydin, 2018). For example consider ρ,

$$\frac{\partial f_{t+1}(x_t)}{\partial \rho} = \frac{\partial f_{t+1}(x_t)}{\partial x_t} \cdot \frac{\partial x_t}{\partial v_t} \cdot \frac{\partial v_t}{\partial \rho}$$

$$= \frac{\partial f_{t+1}(x_t)}{\partial x_t} \cdot \frac{\partial (x_{t-1} - \alpha_t \frac{m_t}{\sqrt{v_t}+\epsilon})}{\partial v_t} \cdot \frac{\partial v_t}{\partial \rho}$$

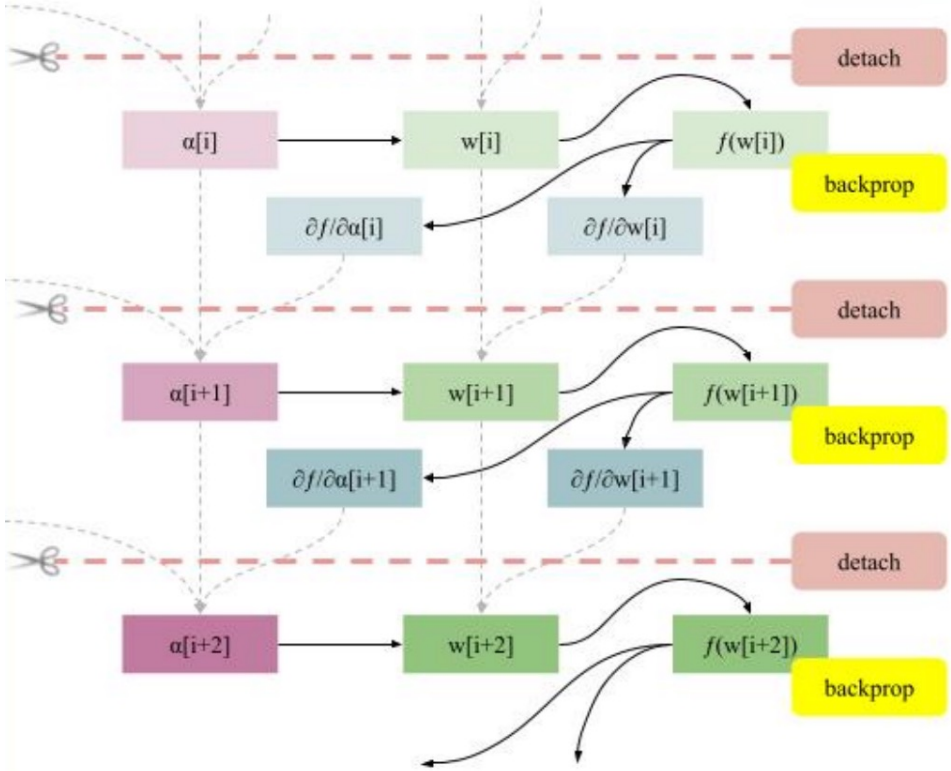
$$= \underbrace{\frac{\partial f_{t+1}(x_t)}{\partial x_t}}_{\text{Already computed}} \cdot \underbrace{\frac{\alpha_t m_t}{2\sqrt{v_t}(\sqrt{v_t}+\epsilon)^2}}_{\text{Manually compute?}} \cdot \underbrace{\left(\tilde{v}_t - v_t^{(max)}\right)}_{\text{Manually compute?}}$$

Hyper-gradient descent

$$\frac{\partial w_i}{\partial \beta_{1i}} = -\frac{\alpha_i \left(-\frac{\partial f(w_{i-1})}{\partial w_{i-1}} + m_{i-1} + i\beta_1{}_i^{(i-1)}\hat{m}_i\right)}{\left(1-\beta_1{}_i^{i}\right)\left(\epsilon_i + \sqrt{\hat{v}_i}\right)}$$

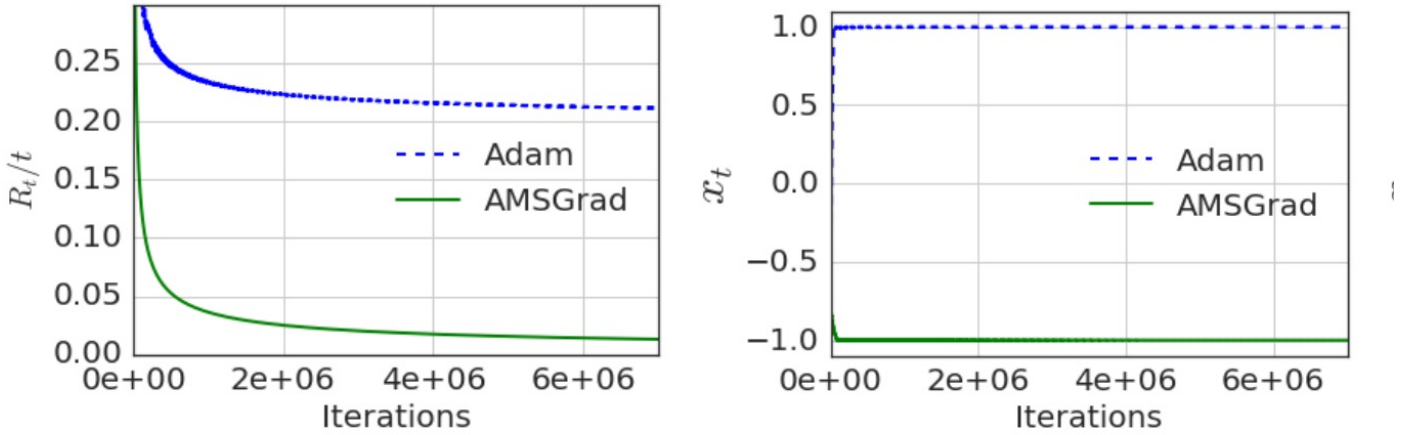
$$\frac{\partial w_i}{\partial \beta_{2i}} = \frac{\alpha_i \hat{m}_i \sqrt{\hat{v}_i}\left(-\left(\frac{\partial f(w_{i-1})}{\partial w_{i-1}}\right)^2 + v_{i-1} + i\beta_2{}_i^{(i-1)}\hat{v}_i\right)}{2v_i\left(\epsilon_i + \sqrt{\hat{v}_i}\right)^2}$$

Manual computation is tedious and not scalable, (Chandra, 2022) embed hyper-gradient computation into Auto Grad.



AVGrad

- AMSGrad was proposed by (Reddi, 2018) to prevent increasing effective learning rate in ADAM, which results in non-vanishing terms in the convergence bounds for ADAM.



- However, it performs worse than ADAM, in general, in practice.

AVGrad

- AMSGrad within MADA decreases the performance. We conjecture the performance drop due to AMSGrad might be because of maximum operator that may block the flow of hyper-gradients.

- We replaced the maximum operator with averaging, observed better performance and remedies the issues addressed by AMSGrad.

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
\bar{v}_t &= \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2 \\
\tilde{v}_t &= \frac{1}{t}(\bar{v}_t + (t-1)\tilde{v}_{t-1}), \\
v_t &= \tilde{v}_t,
\end{aligned}
$$

replaced max operator

AVGrad and an Interpolated Optimizer

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
\bar{v}_t &= \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2 \\
\tilde{v}_t &= \frac{1}{t}(\bar{v}_t + (t-1)\tilde{v}_{t-1}), \\
v_t &:= \rho_t \bar{v}_t + (1-\rho_t)\tilde{v}_t
\end{aligned}
$$

Proposition 1. *The following inequality is a sufficient condition for non-increasing effective learning rate:*

$$\rho_t \leq \frac{1}{t(1-\beta_2)+1}.$$

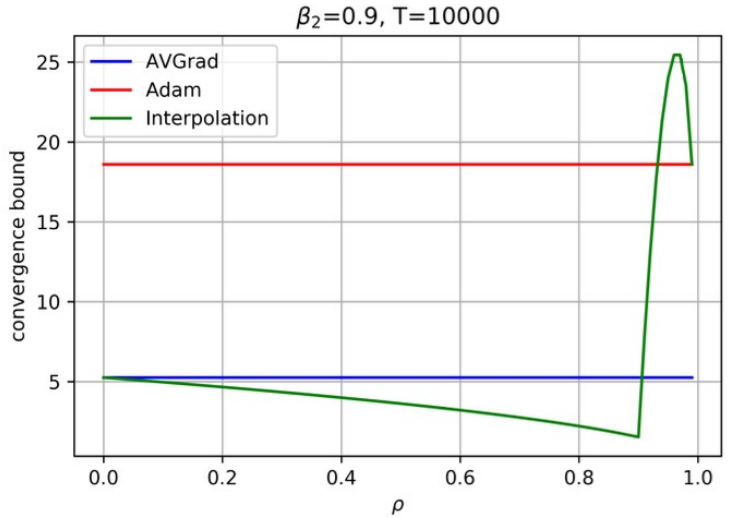
The interpolated optimizer solves the non-increasing learning rate issue as long as the condition holds.

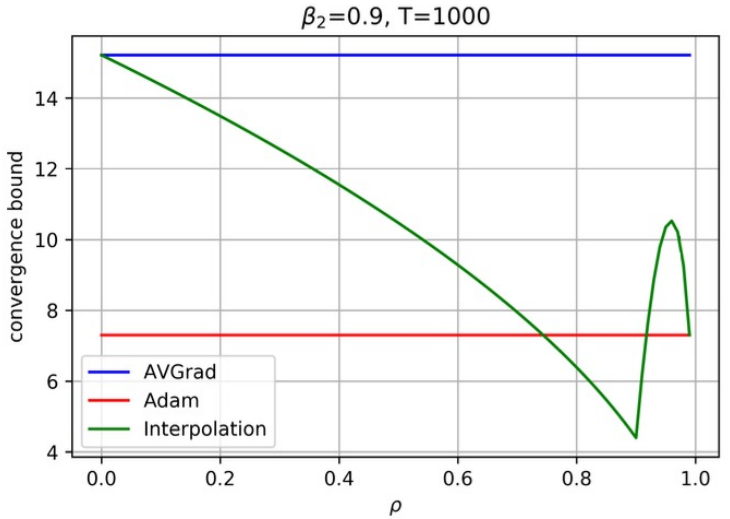
Convergence of the Interpolated Optimizer

Theorem 1 (Convergence of interpolation of AVGrad and Adam without momentum)**.** *Under the assumptions above and* $\alpha_t = \frac{\alpha}{\sqrt{t}}$ *for some* $\alpha > 0$*, and for* $\rho_t = \frac{\rho}{t}$ *for a constant* ρ*:*

$$G_T \leq E(T)\left[\frac{C_1}{T} + \frac{C_2 \ln\left(E(T)\right)}{T} + C_3\left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+\right]$$

where, $G_T = \frac{1}{T}\sum_{t=1}^{T} \|\nabla F(x_t)\|^2$, $E(T) = \frac{\sqrt{\rho+(1-\rho)T}}{\sqrt{1-\beta_2}}$, *and* C_1, C_2, C_3 *are constants independent of* T, ρ, β_2.





Experiments

- GPT-2 (124M, 355M) training on OpenWebText; validation on OpenWebText, Lambada, Wikitext next token prediction.

- Tiny GPT-2 (10M) training on Shakespeare dataset for next character prediction,
- Overall parameterization:

ADAM, Adan, AVGrad, YOGI, LION → Parameterization

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
n_t &= \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1}) \\
m_t^{lion} &= \beta_2^{lion} m_{t-1}^{lion} + (1-\beta_2^{lion}) g_t \\
u_t &= \beta_1^{lion} m_{t-1}^{lion} + (1-\beta_1^{lion}) g_t,
\end{aligned}
$$

$$
\begin{aligned}
\hat{g}_t &= g_t + \beta_3(g_t - g_{t-1}) \\
\tilde{g}_t^2 &= c_{t-1}\hat{g}_t^2 + (1-c_{t-1})(\bar{v}_{t-1} + \hat{g}_t^2 \mathrm{sign}(\hat{g}_t^2 - \bar{v}_{t-1})) \\
\bar{v}_t &= \beta_2 \bar{v}_{t-1} + (1-\beta_2)\tilde{g}_t^2 \\
\tilde{v}_t &= (\bar{v}_t + (t-1)\tilde{v}_{t-1})/t \\
v_t &= \rho \bar{v}_t + (1-\rho)\tilde{v}_t \\
x_{t-1} &= x_{t-1} - \lambda \alpha_t x_{t-1} \\
x_t &= x_{t-1} - \alpha_t \Big(\gamma \frac{m_t + \beta_3 n_t}{\sqrt{v_t} + \epsilon} + (1-\gamma)\mathrm{sign}(u_t) \Big).
\end{aligned}
$$

Experiments

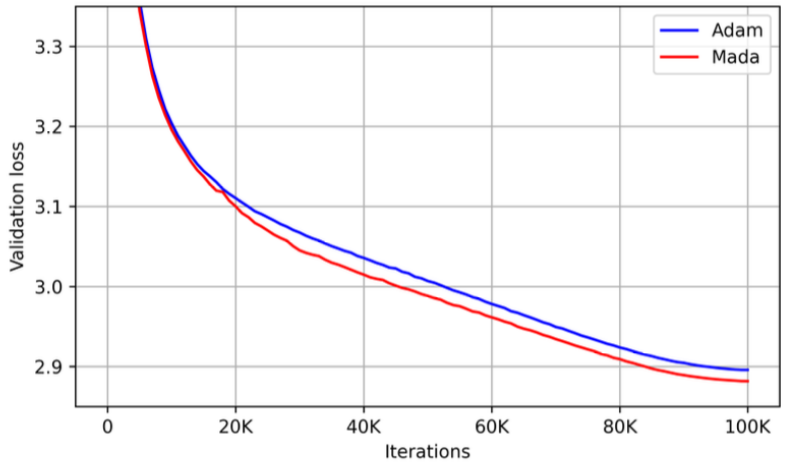


Figure 3: Validation loss on OpenWebText for GPT-2 (125M) model.

Table 2: Validation loss of MADA on OpenWebText vs other adaptive optimizer baselines.

Method	Validation Loss
Baselines	
Adam	2.8956
Adan	2.8896
HyperAdam	2.8950
Lion	2.8892
AVGrad	2.8895
MADA	
MADA	**2.8806**
MADA-FS	**2.8766**
Poor initialization	
MADA$^-$	2.8921
Adam$^-$	2.9157

HyperAdam is the case when only β_1, β_2 are learned

Experiments

Table 3 Validation perplexities of competing methods on OpenWebText, Wikitext and Lambada datasets.

Method	OpenWebText	Wikitext	Lambada
Baselines			
Adam	18.0940	63.8544	77.3314
Adan	17.9863	63.5518	74.6970
HyperAdam	18.0843	61.7717	**72.6803**
Lion	17.9792	61.8661	75.3158
AVGrad	17.9840	64.2620	75.1317
MADA			
MADA	**17.8249**	61.2513	74.2480
MADA-FS	**17.7544**	59.4086	**73.5623**
Poor initialization			
MADA$^-$	18.0317	**57.1613**	75.3550
Adam$^-$	18.4624	72.9017	79.1217

Experiments

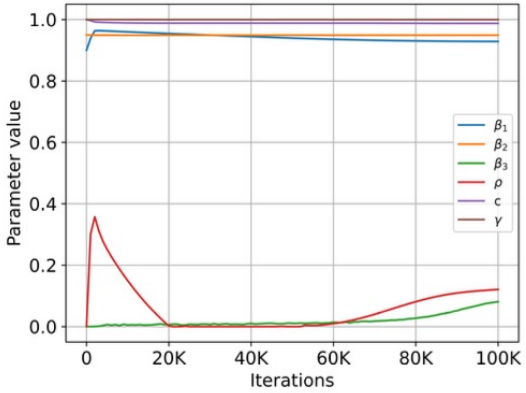


Figure 4: Parameter values with respect to iterations for $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95, \beta_{3,0} = 0$, during training of GPT-2 (125M).

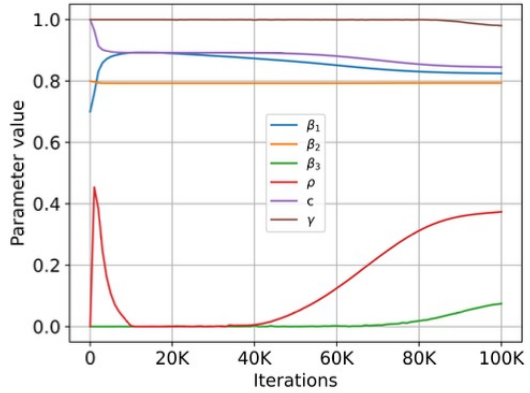


Figure 5: Parameter values with respect to iterations for $\beta_{1,0} = 0.7, \beta_{2,0} = 0.8, \beta_{3,0} = 0$, during training of GPT-2 (125M).

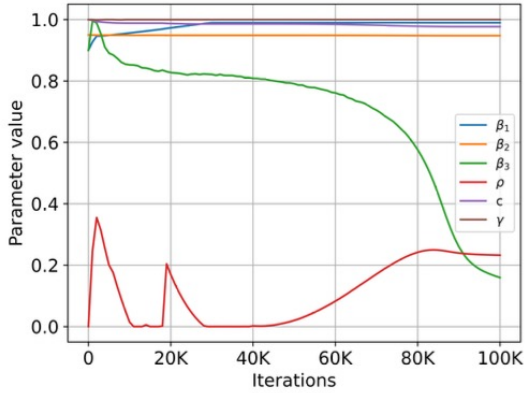


Figure 6: Parameter values with respect to iterations for $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95, \beta_{3,0} = 0.9$, during training of GPT-2 (125M).

Experiments

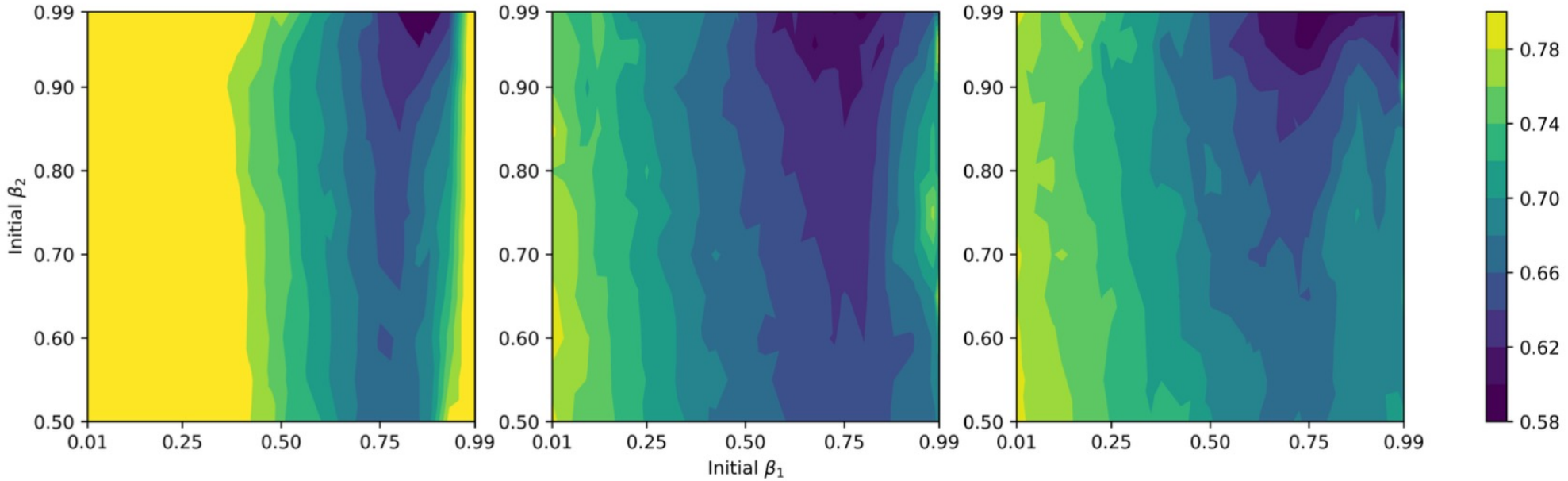


Figure 7: Final training loss of Adam, **MADA**, and HyperAdam vs. initial β values on Shakespeare dataset. **MADA** yields better performance for wider choice of initial β values, illustrating its robustness.

Experiments

Synthetic convex experiment. This is a famous example from [Reddi et al., 2018], where Adam notably fails to converge to the optimal solution, $x = -1$; whereas AMSGrad (and AVGrad) reach the optimum. The online learning experiment given in [Reddi et al., 2018] to motivate AMSGrad is as follows:

$$g_t(x) := \begin{cases} 1010x & \text{for } t \bmod 101 = 1 \\ -10x & \text{otherwise} \end{cases} \tag{32}$$

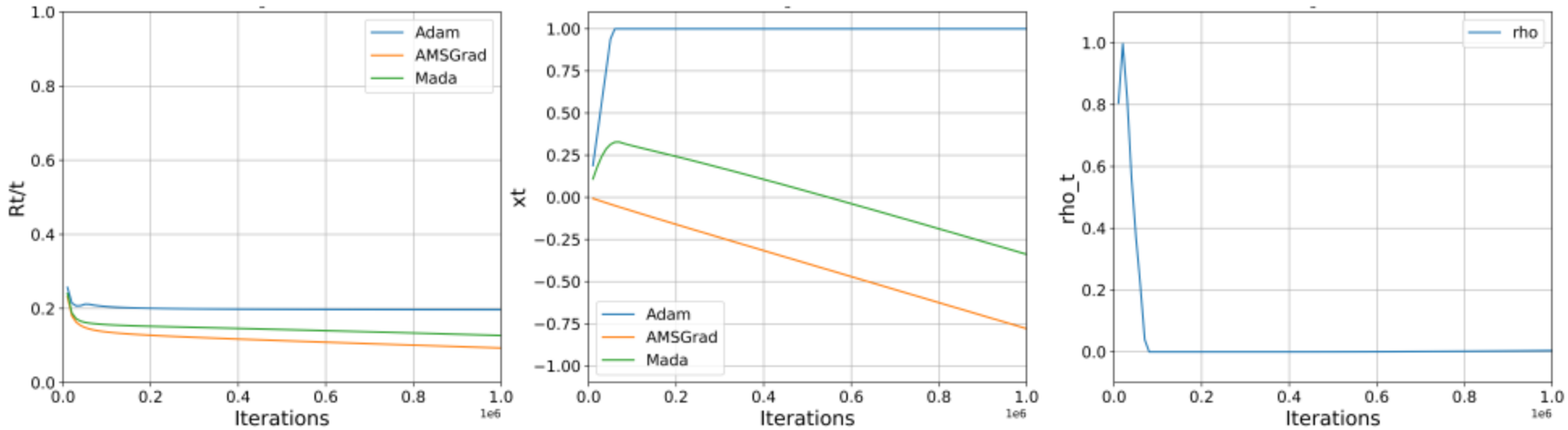


Figure 8: Average regret, x_t, ρ_t with respect to iterations for `MADA` on simple convex function.

Future work

- Find a generic framework for proving convergence of meta-optimizers.
- Add more optimizers into MADA, e.g. LAMB or second order optimizers.
- Extend the formulation to discover new optimizers.

Thank You!

Computational Burden

- If we have T tokens, N model parameters; forward-backward pass requires 6TN FLOPs, update of optimizer parameters require cN FLOPs, where c is between 10-20.
- Computation of hyper-gradients is also O(N), hence overall burden is O(N) and does not depend on T.
- Memory burden is heavier.